CS783 Assignment-1

Amrit Singhal (150092)
Department of Computer Science
IIT Kanpur
Kanpur, India
amrits@iitk.ac.in

Aarsh Prakash Agarwal (150004)
Department of Electrical Engineering
IIT Kanpur
Kanpur, India
aarshp@iitk.ac.in

1 Problem Statement

The problem involves multiclass instance recognition and image retrieval. Looking at the data set, there are 16 different classes of objects. Each class has 216 images associated with covering front and top view at various angles ranging from 0° to 360°. In total, therefore we have 3456 images to be rank in the order of preference to be retrieved in reply to a query.

We have tried many approaches most of them being directed at multi-label classification, which are mentioned below.

2 Vanilla SIFT Feature Matching

Help for this section is taken from tutorial Sif This is the most naive approach taken by us for the problem. Given a test image, we extract the SIFT descriptors of it and try to match those descriptors with the SIFT descriptors of all the training images(one by one) in a specific class, and find the number of 'good' matching descriptors for each image. For finding the 'good' descriptors, we find the nearest two neighbours for each descriptor in test image from the train image, using the Fast library for approximate nearest neighbors apporach as decribed by Muja and Lowe [2014]. If the ratio of distances of two nearest descriptors is less than 0.75, then they are called 'good' descriptors.

If the number of good descriptor is greater than a threshold for a image we call that a 'good' image. Now, for each class, the number of matching 'good' images can be found for a given test image. Hence we can rank the classes in the decreasing order of matching 'good' images found in each class.

The code for this part is contained in the file `vanilla_sift.py`, which can be run `python vanilla_sift.py`. The function `extract_SIFT_features()` is used to obtain the SIFT features for all images, `get_img_matches()` is used to perform the complete process and the third function simply completes the ranking to include the non-matched classes at the end.

3 Visual bag of words approach with SIFT

The next approach used by us is the Visual Bag of Words (VBoW) method using the SIFT feature descriptors. This method relies on getting a representation of each image in terms of basic visual words present all across the corpus, and using this representation to perform classification.

The method starts by getting all the SIFT descriptors across all the training images together. Then, we perform a MiniBatch K-Means clustering to cluster all these descriptors into groups, and obtain a representative cluster center for each of these. Next, for each image, having say K SIFT descriptors, we convert its $K \times 128$ feature representation, into a $K \times 1$ representation, by replacing each descriptor by the cluster id of the cluster to which it belongs. Finally, be perform binning on this vector to get a histogram over clusters, getting a Num_clusters$\times 1$ sized feature vector representation

for each image. These MD5 hash for these stored vectors is available at `image_hist_data.md5`

Having these representation, we now perform ranking/classification using three different methods:

1. Cosine Similarity:
   We calculate the cosine similarity measure of each of the test images with each training image from all classes to obtain a similarity score between them, and we use this similarity score to rank the images in retrieval.

2. MLP Classifier:
   We use these feature vector to train a MLP classifier with Adam Optimizer, using the MLP-Classifier from scikit-learn. The results obtained are available in the folder `mlp_adam_res` and the MD5 hash of the model is available at `mlp_adam.md5`.

3. Random Forest Classifier:
   We use these feature vectors to train a random forest classifier using the RandomForestClassifer from scikit-learn. The results obtained are available in the folder `rfc_model_res` and the MD5 hash of the model is available at `rfc_model.md5`

The code for this section can be found in the file vbow.py. The file contains of multiple functions for the different tasks involved. These functions acan be independently called from the main function, according to the tasks desired to be done.

4 Transfer learning from pretrained Resnets

This experiment was done with the help of Kumar. In this approach our objective is to classify image as one of the many classes it is in by recognizing the classes of objects present in it.

The ResNet-50 or Residual Network is a deep 50 layer neural network which were essentially introduced to solve the degradation and saturation issue and shown to perform well on COCO and Imagenet Datasets. The goal is to learn the residual functions and weights over the pretrained ResNets.

The final model is a sequential with a ResNet-50 stacked with dense softmax layer of 16 perceptrons which takes Image re-sized into 224X224X3 and outputs the probability distribution of image belonging to these classes.

We ran the model for ten, and twenty epochs. Plots for loss and validation accuracy can be see from 1 while training.

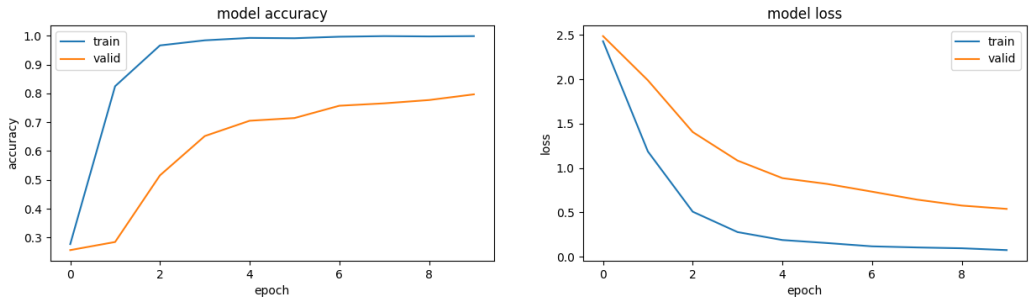

Figure 1: Validation loss and accuracy for 10 epochs

The ResNet model seems to perform decent on the sample test, but the issue is that we get the ranking of the classes and not all the images with in those classes.

5 Bounding Box and Faster R-CNN

One of the major issues in all the above approaches is the role environment plays while training the models especially in cases where SIFT is employed. For the model to focus mainly on the object instead of noise in the surroundings, we have created bounding boxes for the objects. The process is done manually and hence, the bounding boxes are approximate. Basically, you have two point of views for each object and therefore, an approximate bounding box can be calculated based on trial and error basis for each point of view and for each object. The rotation of the object won't affect the bounding box much and therefore each bounding box will be valid for all the angles for a particular point of view. The process is still tedious and harrowing.

The idea was to give these bounding boxes as input to a faster RCNN Ren *et al.* [2015] model and get the job done. Sadly, the Faster RCNN is not so fast while training. For around 3000 images, the necessary epochs should be at least anywhere 1000 to 2000,(while each epoch takes 1000 iterations itself) and epoch was approximately taking around 8 hours on our CPUs. We could not get access to a GPU machine and therefore, this method could not be implemented.

References

Suni Kumar. Tutorial keras: Transfer learning with resnet50 for image classification on cats and dogs dataset. https://www.kaggle.com/suniliitb96/tutorial-keras-transfer-learning-with-resnet50?fbclid=IwAR1McYdSDcaITZxKdUWE7YRMZI9OjjWEIfyqlbpjVn9fI3bcYcFlHNCpop4.

Marius Muja and David G Lowe. Scalable nearest neighbor algorithms for high dimensional data. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, (11):2227–2240, 2014.

Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'15, pages 91–99, Cambridge, MA, USA, 2015. MIT Press.

Open cv tutorial. https://docs.opencv.org/3.1.0/d1/de0/tutorial_py_feature_homography.html?fbclid=IwAR07Qtlt3p6InmDS5F2en-FKdxkr71ktFV_EtHB2Xwm5nYkoFdMBWCnjviw.